COMPUTAÇÃO PARALELA EM JAVASCRIPT

Introdução

Esta pesquisa cientifica tem como ponto de partida a aplicação de conceitos e técnicas de computação paralela na engine de interpretação da linguagem de programação Javascript no web browser Mozilla Firefox.

Tal pesquisa visa identificar os pontos falhos e com lentidão na engine de interpretação de Javascript do web browser Mozilla Firefox. Com isso, avaliar quais técnicas e conceitos podem ser aplicados para obtermos melhor desempenho, velocidade na interpretação dos códigos e melhor aproveitamento dos recursos dos equipamentos nos quais o web browser é executado, sejam eles celulares, notebooks, tablets ou desktops.

Justificativa

- Aperfeiçoar a velocidade de interpretação das engines de Javascript
- Otimizar a utilização dos recursos disponíveis nas plataformas onde os browsers são executados atualmente
- Acrescente demanda por aplicações que rodam no browser (Web Apps)

A utilização de Javascript para tornar essas aplicações ricas, com usabilidade e acessibilidade é quase que mandatório.

Tendo em vista a recente evolução e amadurecimento da linguagem Javascript (que antes era apenas interpretada pelo browser), o surgimento do framework NodeJS, a corrida dos browsers mais populares como Mozilla Firefox, Google Chrome, Apple Safari, entre outros, para oferecer aos usuários e principalmente aos desenvolvedores, engines de Javascript cada vez mais rápidas e eficientes, é fato que a aplicação de técnicas de computação paralela aos recursos das engines de interpretação de Javascript, podem contribuir para o melhor uso dos recursos computacionais por parte dos desenvolvedores, resultando em melhor desempenho das aplicações web e melhor experiência de uso para os usuários.

Lembrando também que temos atualmente um sistema operacional inteiramente criado utilizando ferramentas e linguagens voltadas para a Web como HTML5, CSS e Javascript, chamado Firefox OS.

Nele temos o browser Mozilla Firefox como base para o sistema operacional, ou seja, parte desse sistema é mantido e executado graças a engine de interpretação de Javascript

"IonMonkey" e seu JIT (just in time compiler).

Optamos por utilizar os recursos da engine de interpretação de Javascript e JIT da Mozilla, pois sabemos que todos os códigos estão disponíveis para utilização e alteração (liberdade garantida pela licença de utilização de código da própria Mozilla, chamada MPL) e por sabermos que o projeto sempre esteve aberto a colaboração de voluntários que queiram melhorar a web como um todo.

Problema de Pesquisa

Muitas aplicações feitas para web hoje, demandam de velocidade de processamento na interpretação da linguagem Javascript e poder de rápida renderização do código no lado cliente, ou seja, o browser.

Como temos uma gama de diferentes processadores (single core, dual core, quad core, etc) nos dispositivos, as implementações de código para interpretação e renderização de Javascript nas engines dos browser ainda não é muito bem explorado quanto a utilização de conceitos de computação paralela.

Objetivo

Geral

A pesquisa tem como objetivo implementar e utilizar conceitos de computação paralela na engine de interpretação e renderização de Javascript da Mozilla.

Especifico

A utilização de conceitos de computação paralela, paralelismo, entre outros, será aplicada ao core de interpretação da engine de Javascript da Mozilla chamada "IonMonkey", bem como ao JIT (just in time compiler) que é parte da Javascript Engine.

Com a aplicação desses conceitos teremos ganho na performance da engine, pois utilizando melhor os recursos dos processadores, a interpretação e renderização desse código pelo JIT, será muito mais rápida e com utilização mais eficiente dos recursos da maquina.

Método

Mapearemos na engine de interpretação de javascript, as áreas que necessitam de aplicação de técnicas de computação paralela

Tendo um mapeamento das áreas, a aplicação desses conceitos será feita da seguinte forma:

- Compreensão do código atual
- Compreensão do problema que o recurso causa
- Estudo da melhor técnica a ser utilizada
- Implementação da técnica escolhida
- Testes de funcionalidade
    - Testes funcionais serão escritos utilizando a plataforma Javascript Node.js
- Testes de performance
    - Teste de performance serão feitos utilizando os bundleds NPM (Node Package Manager) disponíveis para a plataforma de Javascript Node.js
- Descrição da solução utilizada

Cronograma

Atividades	**Meses**											
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
Documentação do projeto												
Listagem dos canais de comunicação com comunidade Mozilla												
Ambientação e primeiras compilações dos códigos já existentes												
Analise de códigos que estejam causando lentidão												
Listagem dos possíveis conceitos utilizados para resolução dos problemas												
Implementação dos códigos												
Configuração do ambiente de teste												
Refactoring de código												
Testes de funcionalidade												
Ajustes e correção de bugs												

Referências Bibliográficas

Livros

- FLANAGAN, David. **JavaScript: The Definitive Guide. 6th Edition**, O'Reilly Media,

2011.

- ZAKAS, Nicholas C. **High Performance JavaScript**. O'Reilly Media, 2010.
- STEFANOV, Stoyan. **JavaScript Patterns**. O'Reilly Media, 2010.
- MACCAW, Alex. **JavaScript Web Applications**. O'Reilly Media, 2011.
- WILSON, Jim R. **Node.js the Right Way**. Pragmatic Bookshelf, 2013.
- Trostler, Mark Ethan. **Testable JavaScript**. O'Reilly Media, 2013.
- ZAKAS, Nicholas C. **Maintainable JavaScript**. O'Reilly Media, 2012.
- Bolin, Michael. **Closure: The Definitive Guide**. O'Reilly Media, 2010.
- Burnham, Trevor. **Async JavaScript**. The Pragmatic Programmers, 2012.
- Saleh, Hazem. **JavaScript Unit Testing**. Packt Publishing, 2013.
- Fogus, Michael. **Functional JavaScript**. O'Reilly Media, 2013.

Links

- **CommonJS Specs Wiki** - http://wiki.commonjs.org/wiki/CommonJS
- **Javascript Documentation** - https://developer.mozilla.org/en-US/docs/Web/JavaScript
- **Javascript JIT** - https://wiki.mozilla.org/Modules/All#JavaScript_JIT
- **Running Automated JavaScript Tests** – https://developer.mozilla.org/en-US/docs/SpiderMonkey/Running_Automated_JavaScript_Tests
- **Creating JavaScript tests** - https://developer.mozilla.org/en-US/docs/SpiderMonkey/Creating_JavaScript_tests
- **SpiderMonkey Internals** - https://developer.mozilla.org/en-US/docs/SpiderMonkey/Internals
- **Tamarin** - https://developer.mozilla.org/en-US/docs/Tamarin
- **JavaScript:TraceMonkey** - https://wiki.mozilla.org/JavaScript:TraceMonkey
- **Tracing JIT** - https://developer.mozilla.org/en-US/docs/SpiderMonkey/Internals/Tracing_JIT
- **Nanojit** - https://developer.mozilla.org/en-US/docs/Nanojit
- **Hacking Tips** - https://developer.mozilla.org/en-US/docs/SpiderMonkey/Hacking_Tips
- **SpiderMonkey** - https://developer.mozilla.org/en-US/docs/Mozilla/Projects/SpiderMonkey
- **IonMonkey** - https://wiki.mozilla.org/Platform/Features/IonMonkey
- **About JavaScript** - https://developer.mozilla.org/en-US/docs/Web/JavaScript/About_JavaScript
- **Introduction to the JavaScript shell** - https://developer.mozilla.org/en-

US/docs/SpiderMonkey/Introduction_to_the_JavaScript_shell

- **asm.js** - https://developer.mozilla.org/en-US/docs/Games/Tools/asm.js
- **asm.js AOT compilation and startup performance** - https://blog.mozilla.org/luke/2014/01/14/asm-js-aot-compilation-and-startup-performance/
- **Measuring Code Coverage on Firefox** - https://developer.mozilla.org/en-US/docs/Measuring_Code_Coverage_on_Firefox